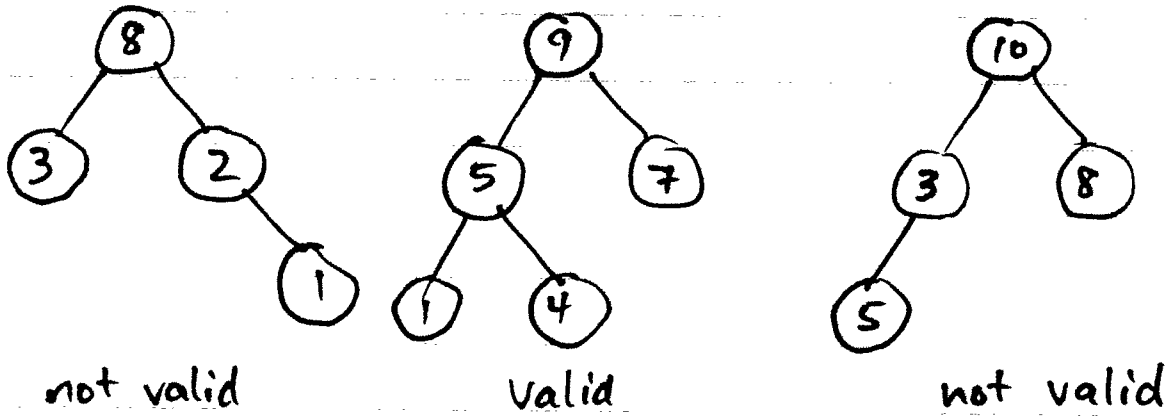
Heaps

- a complete binary tree where the value in each node is larger than the values of all children

Eg:

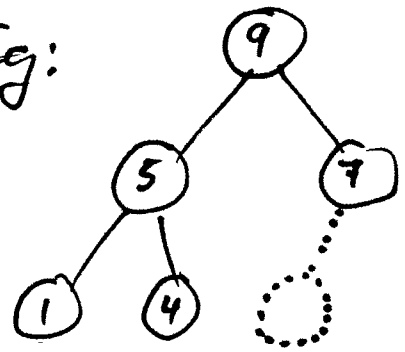


- very useful for implementing a priority queue or for heapsort

Heap Insert

- must insert data and keep binary tree complete and in proper order

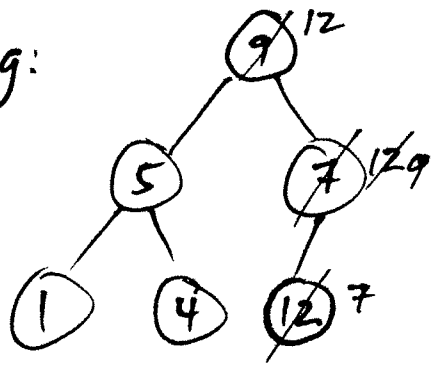
Eg:



- next node must go to right of last leaf on the bottom of tree

- must exchange data values with parent (recursively) until node larger than the new value is found.

Eg:



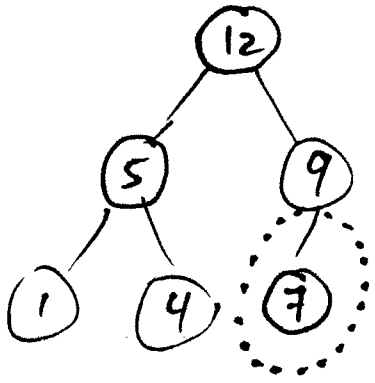
- swap up to root if needed. ($\log_2 N$ steps required)

Heap Delete

- always delete largest value from heap - found at root by defn.

- need to repair heap so tree is complete and nodes are in proper order

Eg:

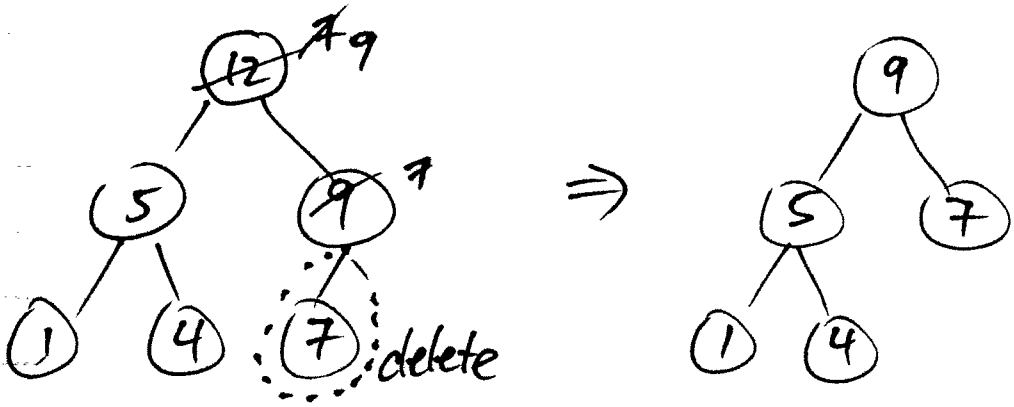


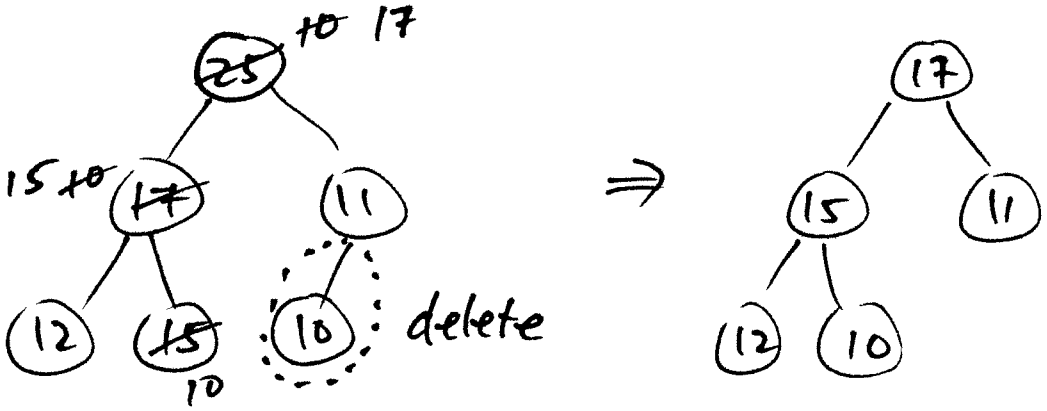
- bottom right node must be removed to stay complete

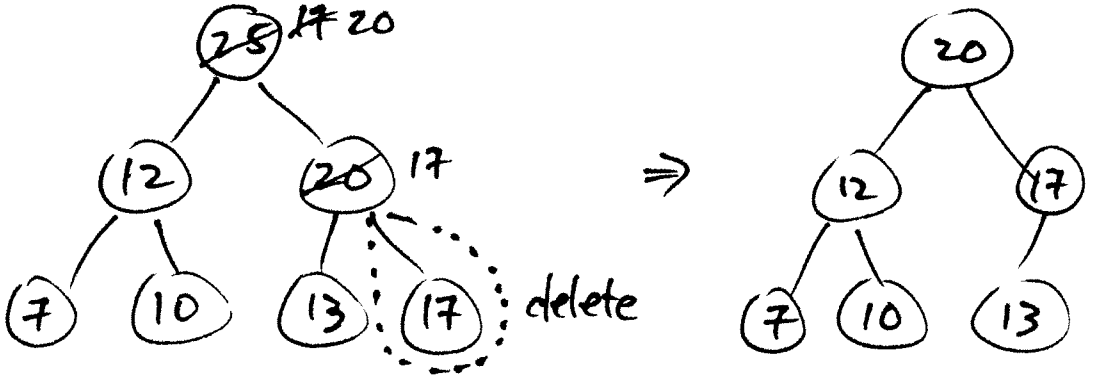
- replace value of root node with value in deleted node and swap values with largest child (recursively) until all nodes are in proper order.

($log_2 N$ steps required)

Delete Examples:

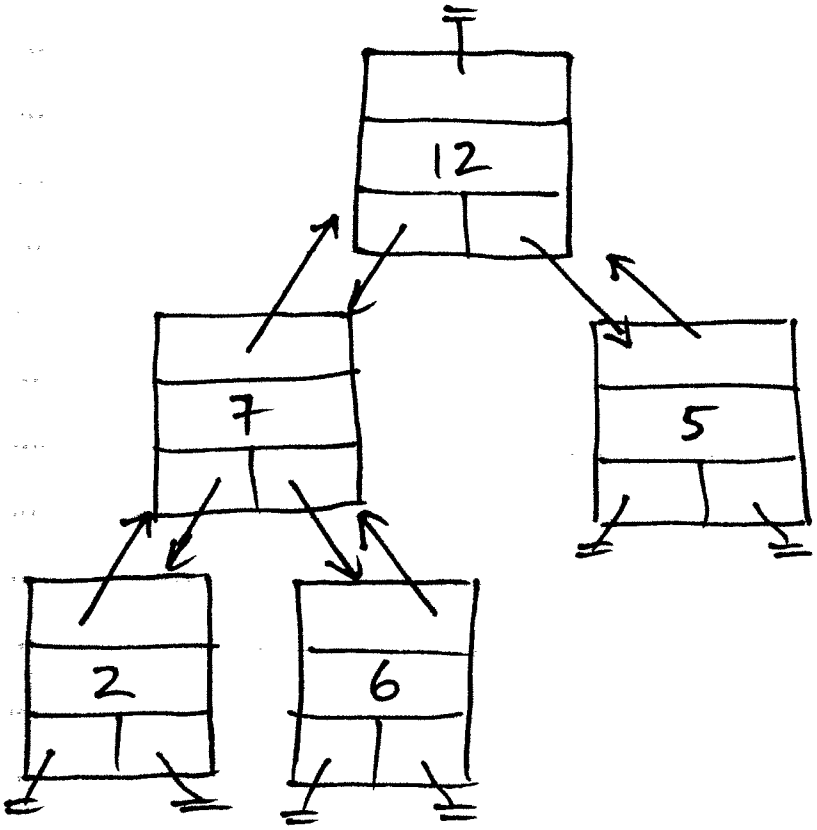






Heap Representations

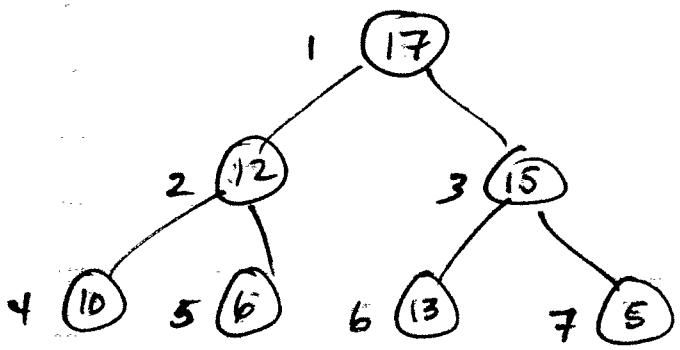
- could define a heap node with value, parent, left, right.



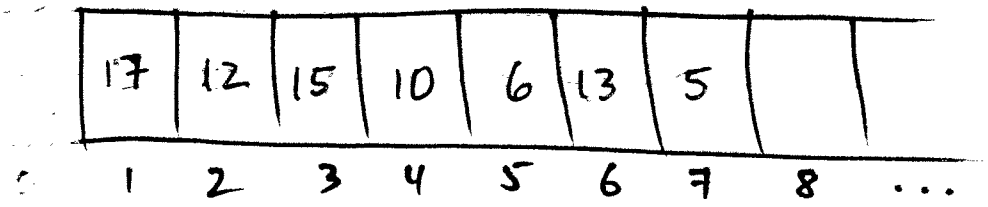
- this would give us a way to go up or down in the tree, but finding the location of node to insert or delete is very painful.

Array Based Binary Tree

- can store any binary tree in an array if we use a node numbering convention to access parent / children.

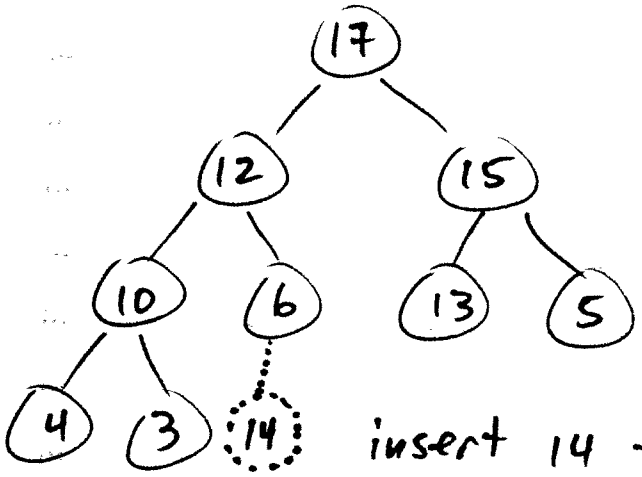


- number nodes from T→B, L→R starting at 1.

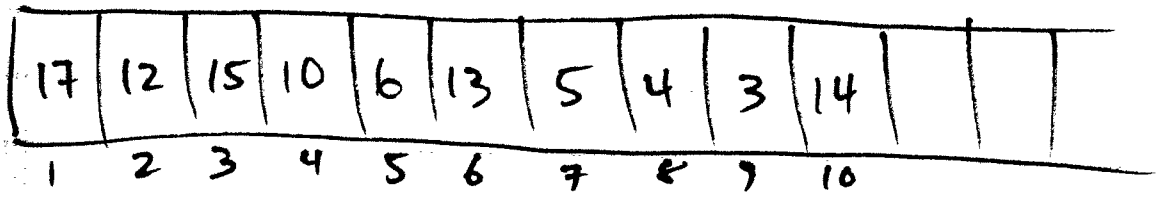


- from any node at position i their parent is at $i/2$, and their left child is at $2i$ and right child is at $2i+1$ ∴ easy to traverse tree.

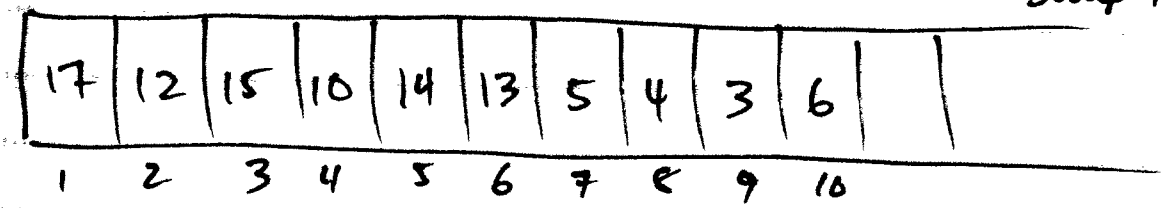
Array Heap Insert



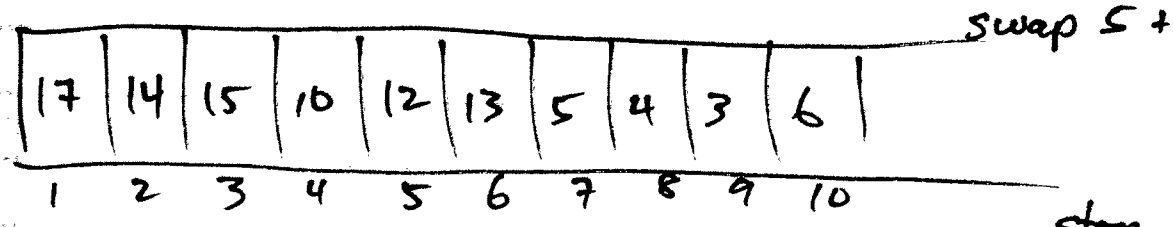
insert 14 to heap



swap 10 + 5

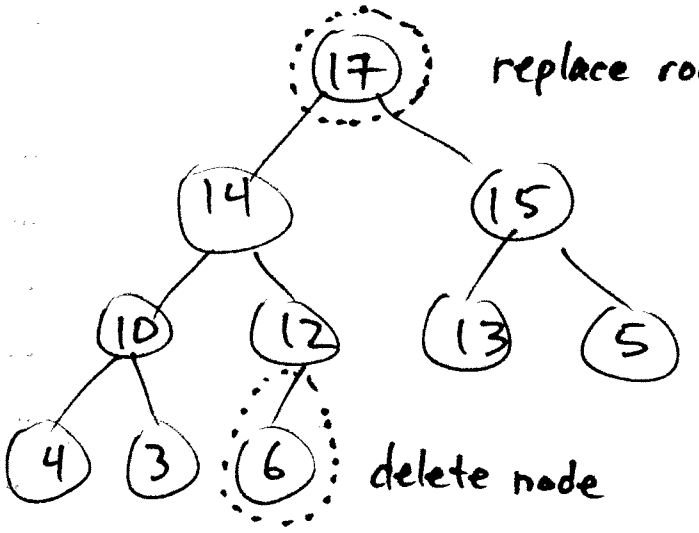


swap 5 + 2



stop.

Array Heap Delete



replace root value with deleted value

delete node

17	14	15	10	12	13	5	4	3	6
1	2	3	4	5	6	7	8	9	10

move 10 to root

6	14	15	10	12	13	5	4	3	
1 ↑	2 =	3 =	4	5	6	7	8	9	

swap 1+3

15	14	6	10	12	13	5	4	3	
1	2	3 ↑	4	5	6 =	7 =	8	9	

swap 3+6

15	14	13	10	12	6	5	4	3
1	2	3	4	5	6	7	8	9

done.

Heap Declaration

```
class heap
{
public
        heap();
       ~heap();

       void insert(int val);
       int delete();
private:
     int size;
     int data[max_size];
};
```

- Assume that the constructor function sets size to zero and puts zeros in data array.

Heap Insert Code

```
void heap::insert(int val)
{
    // shift data down heap
    size++;
    int child = size;
    int parent = child/2;

    while((child > 1) && (data[parent] < val))
    {
        data[child] = data[parent]
        child = parent;
        parent = child/2;
    }

    // insert new value in heap
    data[child] = val;
}
```

- This code makes room for new value by shifting data down the heap.
- Since heap height $= \lceil \log_2(N+1) \rceil$ this step is $O(\log N)$.

Heap Delete Code

```
int heap::delete()
{
    // remove largest value from heap
    int val = data[1];
    data[1] = data[size--];

    // shift data down the heap
    int parent=1;
    int largest=0;
    while (parent != largest)
    {
        // check left
        largest = parent
        int left = parent *2;
        if ((left<=size) && (data[left]>data[largest])
            largest = left;

        // check right
        int right= parent*2+1;
        if ((right<=size)&&(data[right]>data[largest])
            largest = right;

        :
```

More Heap Delete Code

```
        :
        // swap data values
        if ( parent != largest)
        {
            int temp = data[parent];
            data[parent] = data[largest];
            data[largest] = temp;
            parent = largest;
            largest = 0;
        }

    } // end while
    return val;
} // end function.
```

- This code can be modified to store other data types instead of integers as long as comparison ops are available.

- This code is also $O(\log_2 N)$.

Heap Sort

- We can use the property that heap delete always returns the largest value in a set to sort an array of integers.
- The idea is to insert N values into the heap $O(N \cdot \log N)$ steps.
- Next we delete N values (in decreasing size) from the heap to fill output array. $O(N \log N)$ steps.
- Hence heapsort is $O(N \log N)$ like mergesort and quicksort.

Heap Sort Code

```
void heapsort(int data[], int size)
{
    // insert data into heap
    heap h;
    for (int i=0; i<size; i++)
        h.insert(data[i]);

    // remove data from heap
    for (int j=size-1; j>=0; j--)
        data[j] = h.remove();
}
```

- Heapsort can be optimized a little by including the insert / delete code directly in function above.